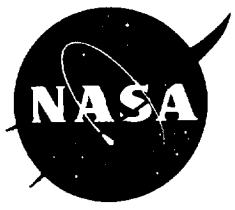
NASA/WVU Software IV & V Facility
Software Research Laboratory
Technical Report Series

NASA-IVV-96-009
WVU-SRL-96-009
WVU-SCS-TR-96-18
CERC-TR-TM-96-009

Specification and Design of a Fault Recovery Model for the Reliable Multicast Protocol

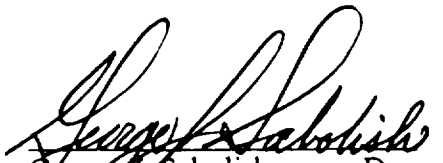
by Todd L. Montgomery, John R. Callahan, and Brian Whetten



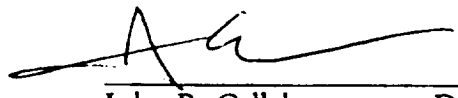
National Aeronautics and Space Administration

West Virginia University

According to the terms of Cooperative Agreement #NCCW-0040, the following approval is granted for distribution of this technical report outside the NASA/WVU Software Research Laboratory



George J. Sabolish Date
Manager, Software Engineering



John R. Callahan Date
WVU Principal Investigator

Specification and Design of a Fault Recovery Model for the Reliable Multicast Protocol*

Todd Montgomery, John R. Callahan, Brian Whetten
{tmont,callahan}@cerc.wvu.edu, whetten@tenet.cs.berkeley.edu
NASA/West Virginia University Software IV&V Facility
100 University Drive
Fairmont, WV 26554
304-367-8235, 304-367-8211(fax)

April 1, 1996

Abstract

The Reliable Multicast Protocol (RMP) provides a unique, group-based model for distributed programs that need to handle reconfiguration events at the application layer. This model, called *membership views*, provides an abstraction in which events such as site failures, network partitions, and normal join-leave events are viewed as group reformations. RMP provides access to this model through an application programming interface (API) that notifies an application when a group is reformed as the result of a some event. RMP provides applications with reliable delivery of messages using an underlying IP Multicast [22, 10] media to other group members in a distributed environment even in the case of reformations. A distributed application can use various Quality of Service (QoS) levels provided by RMP to tolerate group reformations. This paper explores the implementation details of the mechanisms in RMP that provide distributed applications with membership view information and fault recovery capabilities.

1 Introduction

Many distributed programs need to be reconfigured while continuing to provide services, but how a system is reconfigured is often specific to a particular application. Therefore, any application programming interface (API) to a distributed environment should provide an abstract model of reconfiguration because applications will differ in how they handle changes in a distributed environment. For example, teleconferencing applications should be highly resilient to site failures or network partitions because such

*This work is supported by NASA Grant NAG 5-2129 and NASA Cooperative Research Agreement NCCW-0040. More information pertaining to RMP can be found at http://research.ivv.nasa.gov/projects/RMP/RMP.html

failures can be modeled as normal join-leave changes to the group of conference participants. However, distributed database systems that require atomic transactions will be highly sensitive to such failures. In either case, the application must decide what levels of fault tolerance it needs and how to handle changes to other sites and the network in order to continue service.

The Reliable Multicasting Protocol (RMP) [19, 4, 5] is a unique, high-performance protocol developed at West Virginia University in cooperation with NASA that will soon be presented for consideration as an Internet standard and is being used currently in many network software applications. RMP presents an API that provides applications with a simplified model of dealing with complex changes in distributed, group communication environments. RMP provides a programming abstraction, called *membership views*, for handling reliability, resiliency, fault recovery, and ordering issues in a distributed application.

RMP is based on an algorithm originally developed for reliable delivery of data in broadcast-capable, packet-switching networks [7]. The original algorithm allows sites in a packet-switching network to establish a token ring for distributing responsibility for acknowledgments. A single token is passed from site to site around the ring and only the holder of the token (called the current token site) needs to acknowledge certain data packets. RMP has high-performance characteristics because acknowledgments themselves are multicast to all other token ring sites. This approach orders the data packets consistently across all sites and provides a means of passing the token to a new token site.

When a site gets the token (i.e., it becomes the current token site), it multicasts an acknowledgment if and only if it has seen all data packets since the last acknowledgment it received. The token is passed in the multicast acknowledgment packet. The acknowledgment packet includes the source and sequence numbers of data packets it is acknowledging. This allows each site to detect if any packets are missing. A site will use

negative acknowledgments to request retransmission of any missing packets. When all packets since the last acknowledgment received have been received by the current token site, then that site can multicast its acknowledgment and thus pass the token to the next site on the ring. When a token site sends an acknowledgment, it is guaranteed that all data packets since it last held the token have been received by all sites. The sender of a packet assumes that all messages since it last had the token have been received by the other sites within a requested quality of service (QoS) level. A packet is marked delivered if and only if it satisfies its QoS level of delivery. The QoS level allows for resilience of the protocol in the presence of site failures and network partitions. In the case of failures, the token ring reforms itself around the failed site. In the presence of persistent failures, the application program using RMP must decide to degrade the QoS level or try again.

RMP differs from previous reliable broadcast protocols in that an acknowledgment packet may acknowledge an arbitrary number of data packets. Previous protocols specified that each data and acknowledgment packets have a one-to-one relationship. Our approach, however, improves throughput in networks with sporadic losses. Each site in a token ring maintains a data structure called an Ordering Queue (OrderingQ) in which acknowledgments and data packets are organized based on timestamps. An Ordering Queue is consistent if and only if there are no missing data packets for pending acknowledgments. A missing packet will appear as an empty slot in the OrderingQ that must be filled. When a site becomes the token site, all empty slots in the OrderingQ since the last acknowledgment received must be filled. For example, in Figure 1 we show 3 sites of a token ring and a global sequence of events. No site has complete knowledge of this sequence. It is only shown to illustrate a possible scenario. Next to each site is a list of the messages sent by that site. First, site A sends a data packet signified as Data(A,1) where the first parameter is the sending site and the second is the sequence number of the message. Sequence numbers are unique to individual sites. Second, site B sends a

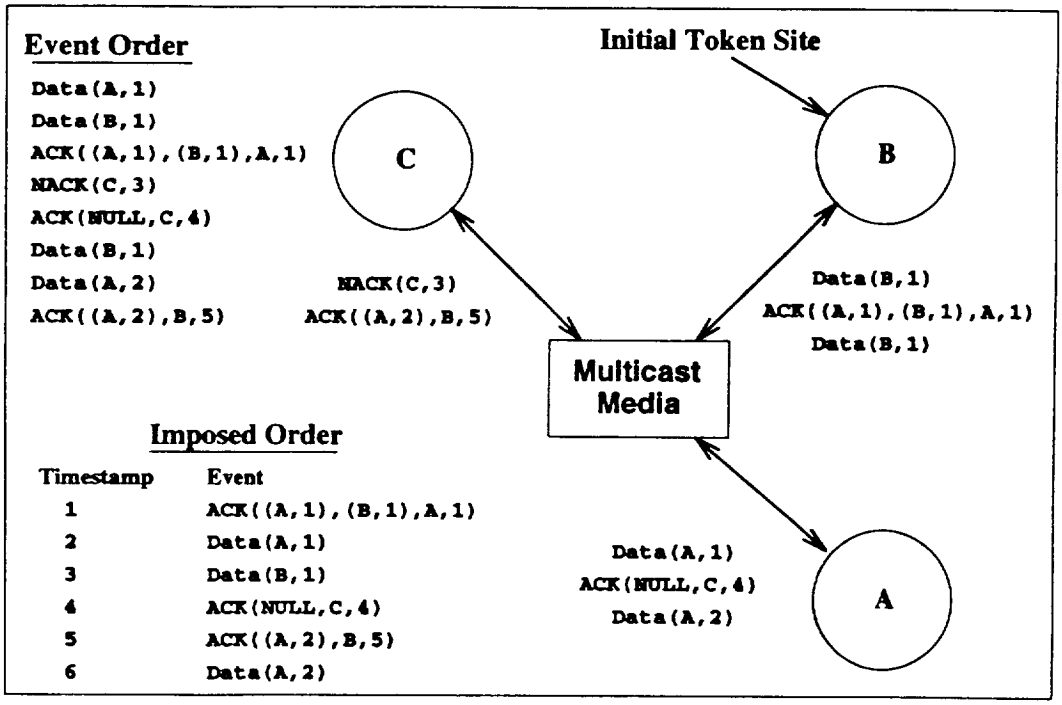


Figure 1: An example of RMP operation

data packet (Data(B,1)). The initial token site is site B who then acknowledges both data packets and passes the token to site A. The Ack((A,1),(B,1),A,1) message contains a list of source identifiers and sequence numbers for two packets, followed by the next token site and the timestamp of the acknowledgment.

In this example, we assume that site C missed the data packet Data(B,1). Site C realizes it has missed a packet after it receives the Ack((A,1),(B,1),A,1) message. It knows this because the Data(B,1) packet is listed in the Ack message from B. Each slot in an OrderingQ corresponds to a timestamp whether explicit in the case of Ack messages or implicit in the case of Data packets. Site C will multicast a Nack message to request the data packet to fill the one slot in its OrderingQ at timestamp 3. Any other site in the ring should respond to this Nack with the requested missing packet. In this example, Site B responds to the Nack by retransmitting the Data(B,1) message. The sequence number identifies this message uniquely to distinguish it from new messages.

If a period passes during which no data packets are transmitted, a site will time-out and subsequently send a multicast NULL Ack packet. In our example, Site A sends a NULL Ack with timestamp 4 after waiting. This NULL Ack passes the token to site C. After the site B retransmits the packet Data(B,1), site A multicasts another data packet with sequence number 2 as Data(A,2). Since site C's OrderingQ is consistent, it multicasts an acknowledgment of the Data(A,2) packet and passes the token to site B. The global ordering of events is an artifact of the timestamps and may or may not reflect the actual order of events. This decentralized notion of ordering, called *global synchrony*, allows applications to synchronize their activities based on group events instead of a single, centralized authority.

2 Background

The basic RMP protocol provides what can be thought of as N-way virtual circuits, called groups, between sets of processes connected by a multicast medium. It is fully distributed, so that all processes play the same role in communication. While primarily using NACKs for error detection and retransmission, RMP provides true reliability and limits the necessary buffer space by passing a token around the members of a group.

RMP provides a wide range of reliability and ordering guarantees on packet delivery, selectable on a per packet basis. In addition to unreliable and reliable but unordered quality of service (QoS) levels, RMP can provide atomic, reliably delivery of packets ordered with respect to each source. It can also efficiently provide delivery of packets in both total and causal order, using causal ordering as defined in [16]. Totally ordered delivery also provides virtual synchrony, as first defined by the ISIS project [23]. Virtual synchrony guarantees that when new members join or leave a group these operations appear to be atomic, so that the sets of messages delivered before and after each membership change are consistent across all sites. Using K-resilient fault tolerance, RMP can provide total ordering and atomicity guarantees even in the face of site failures and

partitions. For a set of packets with a resilience level of K, more than K members of a group have to simultaneously partition away or fail in order to have the possibility of violating the total ordering and atomicity guarantees. By setting K to a number larger than half the members of a ring and not allowing minority partitions to continue, total ordering, atomicity, and virtual synchrony can be guaranteed in the face of any set of arbitrary partitions and failures.

The basic RMP model of communication is a publisher/subscriber model based on textual group names. In the absence of network partitions, any member of a group (a subscriber) will receive all packets sent (published) to the group associated with that group name. RMP also provides a client/server model of communication, where the servers are members of a group and the clients are not members, but can communicate with the servers by sending packets to the group. These packets may be simply acknowledged after being delivered to the group with the requested QoS, or they may be responded to by a single member of the group. RMP uses handlers to guarantee that at most one member will respond to a data packet. Each data packet in RMP has an optional handler number associated with it. These correspond to a set of mutually exclusive handler locks which group members may hold. The group member who holds a given handler lock will be notified upon delivery of a data packet with this handler number that it is supposed to respond to the request. Handler locks are provided in a very efficient way, and can be used for any type of application that requires mutually exclusive locks shared among a group of communicating processes.

A common belief in the research community is that totally ordered reliable multicast protocols are inherently slow. This belief has come about in large part due to the experiences researchers have had with the early versions of ISIS, which for a long time was the only system of this type available. ISIS has since become much faster [2], but the misconception remains. Experience with RMP belies this concept. RMP was tested on 8 SparcStation5's on a 10 Mb/sec (1250 KB/sec) Ethernet. In this environment, the

throughput to a single destination is 1070 KB/sec, or 86% of the network capacity. For group communication to any group of two or more destinations on a LAN, RMP exceeds not only the maximum throughput of TCP/IP, but any other possible non-multicast and non-broadcast algorithm. This is because both the packet latency and throughput of RMP stay roughly constant as the number of destinations increase, whereas the performance of other algorithms decreases linearly. For a group with 8 destinations, RMP has a 7.4 MB/sec aggregate throughput, which is 5.9 times the bandwidth of the supporting Ethernet. The throughput for RMP does not significantly change as a factor of the ordering guarantees, but the per packet latency does. A totally ordered packet will on average have a latency approximately twice that of an unordered or source ordered packet, and this increases for K-resilient packets. This QoS for latency tradeoff is fundamental to distributed protocols, which is why RMP allows this tradeoff to be made on a per packet basis. RMP demonstrates that a fault tolerant, reliable, atomic, fully distributed, totally ordered multicast protocol can actually achieve much better performance in group communication than systems that don't provide these features. For a detailed discussion of the performance of RMP, the reader is referred to [20, 19].

The biggest decision in building a reliable multicast protocol is how to guarantee the reliability and stability of messages without sacrificing throughput or latency. Latency is defined as the time between when a site has a packet to send and when it is delivered to the destination. A message is defined as going stable when the sender knows all destinations have received it. This is the point at which it no longer needs to be held for possible retransmissions. In a reliable multicast protocol, one of factors influencing throughput is the number of ACKs sent per packet, so it is important to minimize this. In order to provide guarantees of total ordering and atomic delivery in the face of failures, a reliable multicast protocol will often delay delivering a packet until after it has received one or more acknowledgments of delivery. This latency for guarantees tradeoff is fundamental to this class of protocols, which is why RMP allows this tradeoff

to be made on a per packet basis.

Traditional protocols use positive acknowledgments (ACKs) from the destination to acknowledge successful receipt of a packet. While quickly providing stability of messages, this approach does not scale well to a multicast system, because each destination has to send an ACK for each packet or set of packets. The use of positive acknowledgements largely defeats the advantage of using multicast packets, because it decreases both the efficiency and the performance of the protocol. Even though positive acknowledgment messages are small, it is because they all are sent simultaneously that they can cause network congestion. In addition, having to process an ACK from each destination increases the load on the sender and decreases the performance of the protocol. One optimization is to not acknowledge every packet. In general, as the number of packets per ACK increases, the length of time for a message to go stable increases, but the lower the load is. As another approach, many systems use negative acknowledgments (NACKs). Negative acknowledgments shift the burden of error detection from the source to the destinations. Packets are stamped with sequential sequence numbers which destinations use to provide reliable delivery by detecting gaps in the sequence numbers and requesting retransmission of the packets corresponding to the gaps. Because the information that a packet has been received is never propagated back to the sender, the senders in these protocols do not ever know for certain that a destination has received a packet. Because of this, senders have to indefinitely keep a copy of each packet sent if the protocol is to be considered truly reliable. In addition, a lost packet will not be detected until another packet is received successfully, which may take a long time if the packet is the last to be sent to the ring for a while. Because of these problems, the RMP algorithm uses a combination of these two approaches. The basic algorithm is based on the ideas of the protocol originally done by Chang and Maxemchuk [8, 7].

The MBusl [6] was the original motivation for RMP. It provides a central server through which clients connect with TCP/IP streams, and an easy to use interface de-

signed to ease the implementation of CSCW applications. It provides both total ordering of messages and reliable multicast, but has very limited scalability, since all packets must be routed through a central point, and duplicate copies sent to each destination.

The Totem protocol [18] is perhaps closest to RMP in its approach, and has reported similar throughput levels to RMP under heavy load. It also uses a rotating token ring approach, but only provides for a single ring for each broadcast domain. Totem avoids using any ACKs by only allowing the current token holder to send data. This provides high throughput under high load over a low latency network, but provides lower throughput and longer latency under low and asymmetrical loads. In addition, because it only allows a single sender to transmit at a time it will provide lower throughput over longer latency networks. To alleviate this problem they have proposed, but not implemented, gateways to link multiple broadcast domains together.

The ISIS system [23, 1] is one of the pioneering protocols in this field. It provides causal ordering and, if desired, total ordering of messages on top of a reliable multicast protocol. The reliable multicast protocol requires separate acknowledgments from each destination, which limits performance. A new system that provides causal ordering on top of IP Multicast has been implemented which is much more efficient than the old system [2], and we are comparing RMP and this new protocol.

The Psync protocol [3] is an ingenious protocol that uses piggybacked ACKs to provide causal ordering of messages and detection of dropped packets. However, both it and the similar Trans [21] and Lansis [15] protocols require that all of the members of the group regularly transmit messages. The Trans protocol and the ToTo protocol [14] implemented on top of Lansis both provide total ordering of messages. These algorithms require that at least a majority of the group members be heard from before a message can be delivered, which causes latency to increase by at least an order of magnitude. For example, for the ToTo protocol to send to a group of 8 destinations under heavy, periodic load from all sources (the best case), the latency is 23.8 ms. This increases to

114.1 ms for lightly loaded poisson sources.

The Multicast Transport Protocol (MTP) [11] is an example of an asymmetric reliable multicast protocol. One site is the communication master which grants "tokens" to group members to allow them to send data. These tokens provide both flow control and total ordering of messages. This causes over dependency on the master, which limits both reliability and performance. MTP also relies exclusively on NACKs for error recovery, which limits reliability and requires extreme amounts of buffer space.

The protocol by Crowcroft and Paliwoda [9] is one of the first protocols to propose reliable multicast over an internetwork which supports hardware multicast. The protocol provides different levels of reliability guarantees, and uses positive acknowledgments from all destinations for reliability. The paper analyzes the flooding problems that occur with simultaneous ACKs from many destinations and proposes a windowed flow control system, in some ways similar to that used in RMP, to alleviate these problems. The xAmp protocol [24] is distributed but also waits for ACKs from all destinations, and so will exhibit performance similar to the earlier versions of ISIS.

The broadcast protocol proposed by Kaashoek et. al. [12] uses a central token site to serialize messages and NACKs for retransmissions. It piggybacks ACKs onto sent messages and has the token site regularly contact silent sites in order to limit buffer space. This protocol has reported very good latency (as low as 1.3 ms for a NULL packet) because it has been implemented on top of bare hardware. However, because each message must be transmitted twice it will fundamentally achieve lower throughput than RMP – 600 KB/sec is a rough upper bound for a 1250 KB/sec Ethernet, as compared to 842 KB/sec for RMP. This will also limit the latency for larger messages; as a 8KB packet in their protocol will spend a minimum of 13.1 ms on the Ethernet, as opposed to 6.7 ms for the message and ACK of RMP.

3 RMP Fault Model

RMP is a modification of a Post-Ordering Rotating Token algorithm originally developed by Change and Maxemchuk[7]. A pass of the token around the ring provides ordering notification to all group members. The token itself acts as a combination of positive and negative acknowledgments to group members for message ordering and reliable delivery without the overhead of large numbers of unicast acknowledgment messages. A message is delivered, or *stable*, if the token is rotated to each of the group members in turn. Once the token has made a complete circuit, it is guaranteed that packets acknowledged previous to the start of the circuit have been received by all group members at that moment.

The actual modifications from the original Chang and Maxemchuk algorithm are quite extensive [20, 19]. Two of the most significant areas of redefinition and extension are in the categories of fault recovery and group membership. Originally, the algorithm only dealt with steady state operation and a very restrictive fault recovery process. i.e. no attention was played to changing the number of members during operation or of relaxing the fault recovery process to allow applications with less stringent requirements to continue operation. RMP expanded this by adding the ability to change a groups membership dynamically so that members can join and leave a group, integrating this ability into the protocol operation smoothly, and using the concept of *membership views* to adjust the fault recovery process on an individual group basis.

3.1 Membership Views

Applications using RMP will receive asynchronous events from RMP layer that indicate delivery of messages, some exceptional conditions being met, or a change to the group in some way. A *membership view* is a snapshot of a group's current membership information that is passed up to the application. This snapshot is part of the globally ordered sequence of events that all group members perceive. All group members receive

the same sequence of events, both messages and membership views, regardless of the underlying event sequence imposed by an unordered and unreliable network. The membership view concept allows RMP to provide a *virtually synchronous* execution model to applications using it. Virtual synchrony was defined by Ken Birman from his work on the ISIS system[23, 1]. "Intuitively, this means that the user can program as if the system scheduled one distributed event at a time"[23]. This approach greatly simplifies distributed application development and provides a convenient service upon which configurable systems can be built. The original Chang and Maxemchuk algorithm fails to provide virtual synchrony due mainly to its lack of membership changes, however, the algorithm also violates virtual synchrony by allowing members to be added during the fault recovery process.

A change in the membership view is an event that returns the new membership view and notifies the application as to the type of event that took place. Some of the more interesting and useful membership view change types are:

- A member has been added to group (or formed own group)
- A member has been removed from group
- A member received a lock [1]
- A member was denied a lock
- A member released a lock
- A member changed the Minimum Size Requirement (MSR) (see Section 3.3)
- Some other member change occurred (add, remove, lock change, etc.)
- A fault was detected and recovery is complete
- Group scoping was changed (i.e., a change in the IP Multicast Time-To-Live (TTL) field)[2]

When a membership change occurs, an application is notified that a group change has occurred, what kind of operation occurred (join or leave), and the status of the group after the change. The change can be categorized into three classes. First, a change may be a *local* change that affects only the notified member. Local changes are

[1] RMP provides 256 mutually exclusive locks for members to use.

[2] RMP uses the IP Multicast scoping mechanism of Time-To-Live (TTL) for controlling the propagation of RMP multicast traffic to group sites on a Wide-Area Network such as the Internet.

the results of requests such as asking to join a group, asking to be removed from a group, or requesting a change to a lock. *Remote* changes are changes that affect other sites. These include local changes to other group members, but to an individual application the changes appear to be remote. Finally, *global* changes affect more than one member of a group. These are changes such as change of group scoping, notification of fault recovery completion and the result of the recovery process.

RMP delivers a membership change event to the application upon the completion of the fault recovery process. This process, called *reformation*, may be successful or unsuccessful depending on extent of site failures, partitions, or leave events. At the end of the reformation process, the result of the reformation is delivered as an event to the remaining group members. Thus an application can examine the membership view and the result of the fault recovery process in order to decide what actions it must take to remain operational. In addition to notification of a fault, RMP allows the application to specify message resiliency on a per message basis as well as allowing each member to have a "vote" on the minimum size of a group to be allowed to proceed after a failure.

3.2 RMP Failure Assumptions

Key to any fault recovery and detection system is defining under what circumstances and assumptions the system is assumed to operate. The result of any fault recovery operation can be: (1) a success, (2) an atomicity violation, or (3) a failure. An *atomicity violation* occurs when the fault recovery process can not attain a common sequence of events between members of a group. In practice this situation is very rarely encountered, but it is possible. Atomicity violations can occur because causally related events may become misordered due to buffering or Internetworking constraints. RMP makes three assumptions pertaining to failures. These are:

- A *site failure* means the site stops processing. The site does not interject corrupting information into a group.

- A *message failure* can be the result of an overly full buffer at either the receiver or the sender, or it may be the result of a transmission failure. ($\ll$ 1% of packets on current local and wide-area networks)
- A failure is detected by a group when communication with the group and a site fails after R attempts. R must be chosen such that a failure is mistakenly detected infrequently, but large enough to provide timely notification of actual failures.

Additionally, RMP addresses the first assumption by supporting cryptographic authentication. This does not completely remove the assumption, but it provides a mechanism whereby corruptive sites can be filtered *if* they can be detected. This method also provides protection from unknown sites that may try to corrupt RMP operation. However, this approach is only as secure as the means by which the members can retrieve the authentication keys and the trustworthiness of the other mechanisms involved.

3.3 Fault Detection

As mentioned above under the failure assumptions, RMP performs failure detection using a series of retransmissions of messages. If a certain amount of retransmissions are attempted without a reply being seen, then the fault recovery process is initiated.

A duality between flow control and fault detection exists that is important to mention. RMP's flow control mechanism uses a slight modification of an adaptive flow control scheme [13]. This scheme dictates retransmissions rates and timeouts between retransmissions to avoid saturating the network during congestion. The adjustment in retransmission periods has a direct bearing on fault detection in RMP. This aspect of RMP operation is continually undergoing experimentation and analysis, however preliminary experiments have shown that an R value set to 10 (for 10 retransmission attempts), and capping the maximum retransmission period to 2 seconds provides timely notification on a LAN [3]. In WAN environments, the R value must be set higher (to well over 30 or more). Changes to the maximum retransmission period for WAN groups

[3] 10 attempts at 2 seconds per attempt implies a maximum detection time of 20 seconds

have shown that 2 seconds [4] works best as long as the maximum packet sizes are also kept small in order to reduce, or eliminate, fragmentation. Higher amounts of fragmentation increase the likelihood of a packet being dropped due to a segment being lost. Currently, RMP can adjust the R value based solely on the group scoping value (i.e., the IP multicasting packet TTL value). Thus it is easy to determine what R value to use based on whether the RMP group stretches over multiple LANs or is based on a single LAN. Other adaptive schemes could also be used to dynamically configure the R value based on previous attempts and other flow control variables. However, this approach must be carefully examined so that the R value does not grow too large to make fault detection times too large [5].

3.4 Selection of Resiliency and Fault Tolerance Levels

An RMP application may choose message ordering and resiliency semantics on a per message basis. These semantics are defined as RMP *Quality of Service (QoS)* values that range from unreliable and unordered to totally ordered and totally resilient. RMP QoS is organized into a hierarchy that begins with varying levels of ordering and progresses into resiliency. Message resiliency is based on assurances that RMP places on how many group members have received a given message using properties of the protocol operation. In order to meet any resiliency guarantees, the message must also meet total ordering guarantees. Thus all resilient messages are, by definition, totally ordered. K resiliency assures that K members of a group receive the message. The value of K may range from 1 to the size of a group, N. Majority resiliency is the special case where $K = \lfloor N/2 \rfloor + 1$, and total resiliency corresponds to the case where $K = N$.

In RMP's execution model, message delivery and message resiliency are separate causally related events. Message delivery is based on ordering alone, while resiliency

[4] 30 attempts at 2 seconds per attempt implies a maximum detection time of 60 seconds

[5] As some would say happens in TCP.

is based on the number of token passes after total ordering is met. This separation of delivery semantics from resiliency notification allows applications to design efficient transaction and persistent object systems. In addition, RMP allows the application to request notification when a message has become stable. This notification is another event that the application may use to help facilitate its operational correctness with respect to group consistency.

When a new member is added to an RMP group, the member, in effect, casts a vote for the minimum size it requires to be maintained after a failure. The actual minimum size of a group is the maximum of the votes from all members. This *Minimum Size Requirement (MSR)* determines the fault tolerance level used by RMP during a reformation. A member may change this vote at any time during normal operation. Such a change is a change to the membership view and the application is notified of this change. The levels of fault tolerance closely reflect the levels of message resiliency discussed above. It is highly desirable for an application to use message resiliency and a specific fault tolerance level to its advantage to provide assurances it may need, i.e. majority resiliency combined with majority fault tolerance assures that if a fault is recoverable, then someone in the group has the message if its resiliency was met. The selectable levels of fault tolerance are:

- *K Fault Tolerance* - Up to $N - K$ members may fail, N is the group size.
- *Majority Fault Tolerance* - Up to $\lfloor N/2 \rfloor - 1$ members may fail. Defining how this majority is calculated may be done in one of two ways [6]:
  - Optimistic - $\lfloor N/2 \rfloor + 1$, where N is defined as the number of members currently in the group.
  - Pessimistic - $\lfloor N/2 \rfloor + 1$, where N is defined as the number of current members plus members who have left since the last stability point.
- *No Fault Tolerance* - No members may fail.

If the desired fault tolerance level is not met after a reformation, then the reformation is classified as a failure and the application is notified. At this point, the application

[6] Currently, RMP uses the optimistic method, however, a formal proof is still underway to determine if this is correct or not.

must then decide how to re-form or re-join the group and continue operation. A common scheme for doing this is to use a logging facility to synchronize group members to a specific point that they all agree upon, re-form or re-join the group, and then continue operation.

4 Fault Recovery Process

The original Chang and Maxemchuk algorithm [7] presents a very high-level and restricted reformation process that is not very applicable in many domains. RMP expands on this by relaxing some requirements, specifying the algorithm using state tables [4, 5], and accounting for other RMP features, such as Multi-RPC, security/authentication, and dynamic membership changes. RMP does not allow members to be added through reformation. This was allowed in the original algorithm, however it violates virtual synchrony.

The fault recovery process **must** terminate and be free of livelock. This property is absolutely critical for continuous operation especially when changes occur during the reformation process itself. At each stage of the reformation process, secondary point failures must be detected. In the state model this is done by using the normal fault detection methods on the fault recovery messages and/or providing a timeout so that the state is eventually changed. During the fault recovery process, the members of the group attempt to come to a common *synchronization point* that indicates the ordering that the fault will take in the global ordering of events. This point must be after all events that have already been ordered by all members. This ordering allows faults to actually be seen as just other events that occur and can be taken into account by the application.

4.1 Two-Phase Commit

The RMP fault recovery process is a *Two-Phase Commit* procedure. The member of a group that first detects a fault is called the *Reform Site* for that reformation. The reform site is responsible for coordinating and moderating the reformation process. The other members of the group are then classified as *Slave Sites*. Slave sites are passive and reactive participants in the reformation process. The two phases are described below.

In Phase 1, the Reform Site multicasts notification of failure to the group while the Slave Sites unicast their responses to the Reform Site. These responses indicate a synchronization point and desire to participate in the reformation process. The Reform Site then determines the membership view for the reformation. This will consist of a subset of the set of sites previously in the group before the fault was detected, i.e. if S is the set of sites before the fault, then $S' \subseteq S$, where S' is the set of sites after the fault. The sites not in S' are sites that are considered to be dropped. The Reform Site then determines the synchronization point common to all members of the group. If this point is not reachable then an atomicity violation has occurred. If the MSR for the group is met and an atomicity violation did not occur, then the membership view is defined to be *valid*, otherwise, the view is assumed to be *invalid*. Thus an atomicity violation indicates an invalid view regardless of meeting MSR or not.

In Phase 2, the new membership view is installed at the surviving sites. The Reform Site multicasts the membership view and the Slave Sites unicast their response to acknowledge reception of the membership view. The Reform Site receives confirmation from all reformation participants of reception of membership view. If the new membership view is valid, then all the sites return to normal operation once reception of all confirmations is received and notify application of fault and successful fault recovery. If the membership view is invalid, then all sites return to the RMP "idle" state and notify the application of the fault and that fault recovery failed.[7]. If the membership view is

[7] In effect, this will disband the RMP group.

valid and confirmation from one or more members does not arrive within a retransmission cycle of the membership view, then the reform site assumes a secondary failure, aborts the current reformation, and the process begins again.

The process is optimized so that when a failure is detected erroneously, RMP does not spend vast amounts of time processing needless information. This optimization is performed by short-circuiting some steps if all sites are heard from. Even more drastic levels of optimization could be performed if the fault cause could be isolated. However, this is very difficult to perform generically with RMP.

4.2 Aborting a Reformation

In some cases it is necessary to abort the current reformation process and begin again. This is performed in cases where multiple reformations are detected, or a secondary failure is detected. Multiple reformations can be detected by the Reform or Slave sites when fault notification is originated by members other than the Reform Site. Secondary failures of the Reform Site can be detected by the slave sites through the use of retransmission cycles for the responses they unicast to the Reform Site. Secondary failures of slave sites can be detected through retransmission cycles used in installing the membership view by the Reform Site.

When a Reform or Slave Site sees a condition that suggests that reformation should be aborted, that site then enters an *Abort Reformation* state and notifies the group to abort the current reformation. At any time during reformation, if a site sees this type of notification, then it must also enter the *Abort Reformation* state.

Members that enter the abort reformation process set a random timeout so that deadlock in the process is avoided. The first site to have this timeout expire, then becomes the new reformations reform site and starts the reformation process. While in the abort reformation process, if a site detects a new reformation beginning, it then

participates as a slave site [8].

4.3 Return to Normal Operation

Returning to normal operation is of high importance to any fault tolerant system. RMP allows operation to continue in all cases, but the application must examine the result of the reformation to assess the correct behavior. This may mean reforming a group and rejoining its members (in the case of atomicity violations and failed reformations), or continued processing (in the case of successful reformations).

Because RMP allows the application to specify a desired MSR, cases can arise where network partitions can cause multiple groups to partition away and continue operation. Once this occurs, RMP operation does not allow the new sub-groups to rejoin if the network is repaired. This is achieved through the same mechanisms that RMP uses to allow multiple groups to exist on a given multicast address. Each packet contains an identifier that explicitly identifies that packet to belong to a specific group. This identifier is called a *Token Ring ID*, or TRID. A TRID is a triple guaranteed to be unique in space (IP address and UDP port) and time (12 hour epoch timer). The TRID is changed on a regular basis, i.e. every 45 minutes, and is also changed for every attempted reformation. Thus two reformations that occur on a partitioned network can be filtered based solely on TRIDs. Allowing partitions to come back together can more easily be done at a higher level than RMP. However, some work with other reliable broadcast/multicast protocols have produced interesting methods of rejoining partitions [15, 18]. Other methods of filtering have also been suggested [17]. It is our belief that applications can benefit from these works to expand RMP's fault recovery process to include successful recovery from atomicity violations and the rejoining of partitions.

[8] Several details here are elided for brevity, including using version numbers for membership views to determine viability of reformations. The full details are given in [20, 25, 26]

5 Conclusions and Future Work

RMP provides a mechanism for continuous, reliable, fault-tolerant, and atomic delivery of messages in a multicast media even in the event of site failure, network partitions and normal join-leave events. In addition, RMP provides an event-based API that presents the application with a powerful and intuitive distributed programming model. This model allows the application to make educated decisions about dynamic group reconfigurations of the application. RMP's fault recovery mechanisms allow the application to tailor itself to any desired level of fault tolerance and message resiliency without mandating that the application explicitly perform these functions itself.

Several important issues remain to be investigated including the possibility of continued operation using the group after an atomicity violation, abstractions for defining semantics of a "majority" (pessimistically or optimistically) for an application, and effective flow control that is orthogonal (or at least alternatively complementary) to fault detection. Isolation of faults in order to optimize the fault recovery process seems to hold promise, but RMP's operation model allows special cases to exist where this issue becomes very difficult to tackle effectively.

References

[1] K. Birman. The Process Group Approach to Reliable Dsitributed Computing. *Communications of the ACM*, 36(12):37–53, December 1993.

[2] K. Birman and T. Clark. Performance of the Isis Distributed Computing Toolkit. Technical Report TR-94-1432, Cornell University, December 1994.

[3] L. L. Peterson N. C. Buchholz and R.D. Schlichting. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–246, August 1989.

[4] J.R. Callahan and T. Montgomery. Verification and Validation of a Reliable Multicast Protocol. In *Proceedings of the 2nd Safety Through Quality Conference*, pages 83–96, 1995.

[5] J.R. Callahan and T. Montgomery. Approaches to Verification and Validation of a Reliable Multicast Protocol. In *Proceedings of the International Symposium on Software Testing and Analysis*, 1996. (to appear).

[6] A. Carroll. *ConversationBuilder: A Collaborative Erector Set.* PhD thesis, Department of Computer Science University of Illinois, 1993.

[7] J. M. Chang and N. F. Maxemchuk. Reliable Broadcast Protocols. *ACM Transactions on Computer Ssystems*, 2(3):251–273, August 1984.

[8] J.M. Chang and N.F. Maxemchuk. A broadcast protocol for broadcast networks. In *GLOBCOM '83*, December 1983.

[9] J. Crowcroft and K. Paliwoda. A multicast transport protocol. In *ACM SIGCOMM '88*, pages 247–256, 1988.

[10] S. Deering. Host Extensiosn for IP Multicasting. Technical Report RFC-1112, IETF, August 1989.

[11] S. Armstrong A. Freier and K. Marzullo. Multicast Transport Protocol. Technical Report RFC-1301, IETF, February 1992.

[12] M. F. Kaashoek A. S. Tanenbaum S. F. Hummel and H. E. Bal. An efficient reliable broadcast protocol. *Operating Systems Review*, 23(4):5–19, October 1989.

[13] V. Jacobson. Congestion Avoidance and Control. In *SIGCOMM Proceedings*, pages 314–328. ACM, 1988.

[14] D. Dolev S. Kramer and D. Malki. Early delivery totally ordered multicast in asynchronous environments. In *23rd Annual International Symposium on Fault-Tolerant Computing (FTCS)*, pages 544–553, June 1993.

[15] Y. Amir D. Dolev S. Kramer and D. Malki. Transis: A Communication Sub-system for High Availability. Technical Report CS9113, Hebrew University of Jerusalem, November 1991.

[16] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, July 1978.

[17] S. Floyd V. Jacobson C. Liu S. McCanne and L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. In *SIGCOMM Proceedings*, pages 342–356. ACM, August 1995.

[18] D. Agarwal P. Melliar-Smith and L. Moser. Totem: A Protocol for Messaging Ordering in a Wide-Area Network. In *First ISMM International Conference on Computer Communications and Networks*, pages 1–5, June 1992.

[19] B. Whetten T. Montgomery and S. Kaplan. A High Performance Totally Ordered Multicast Protocol. In *Theory and Practice in Distributed Systems*, number 938 in LCNS. Spring Verlag, 1994.

[20] T. Montgomery. Design, Implementation, and Verification of the Reliable Multicast Protocol. Master's thesis, West Virginia University, December 1994.

[21] P. M. Meillar-Smith L. E. Moser and V. Agrawala. Broadcast protocols for distributed systems. *IEEE Transactions on Distributed Systems*, 1(1):17–25, January 1990.

[22] D. Waitzman C. Partridge and S. Deering. Distance Vector Multicast Routing Protocol. Technical Report RFC-1075, IETF, November 1988.

[23] K. Birman A. Schiper and P. Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, August 1991.

[24] Verissmo. xamp: A multi-primitive group communications service. In *Proceedings of the 11th Symposium on Reliable Distributed Computing*, 1992.

[25] T. Montgomery B. Whetten and J.R. Callahan. The Reliable Multicast Protocol Specification: Protocol Operation. Technical Report NASA-IVV-95-003, NASA/WVU Software IV&V Facility, 1995.

[26] T. Montgomery B. Whetten and J.R. Callahan. The Reliable Multicast Protocol Specification: Protocol Packet Formats. Technical Report NASA-IVV-95-004, NASA/WVU Software IV&V Facility, 1995.